Swarm Drone Simulation Hackathon
Using Crazyswarm and pycrazyswarm (Simulation Only)

ROBOTECHNITK

March 6, 2025

Contents

1 Introduction

Welcome to the Swarm Drone Simulation Hackathon! In this competition, you will develop innovative applications to control a swarm of Crazyflie drones using the `pycrazyswarm` Python API and ROS in simulation mode only. Your solution will be built in five stages:

Stage 1: Basic Flight (Single Drone): Initialize the simulation and execute a simple flight sequence (takeoff, hover, and land) for a single drone.

Stage 2: Trajectory Tracking (Single Drone): Command a single drone to follow a smooth trajectory between points.

Stage 3: Coordinated Flight (Multiple Drones): Synchronize multiple drones to take off, form a formation, and land.

Stage 4: Dynamic Trajectory Control: Implement dynamic re-planning by having a drone follow a moving target (e.g., a circular path).

Stage 5: Advanced Swarm Behavior with Collision Avoidance: Combine multi-drone coordination with collision avoidance in a dynamic formation.

All development is done in simulation mode. Remember to build your ROS workspace and source it before running any scripts:

```
source ros_ws/devel/setup.bash
```

2 Stage 1: Basic Flight (Single Drone)

2.1 Objective

Initialize the simulation and perform a basic flight sequence: takeoff, hover, and land with a single Crazyflie.

2.2 Boilerplate Code

```
#!/usr/bin/env python
"""
Stage 1: Basic Flight Control for a Single Drone
This script initializes the Crazyswarm simulation and commands one drone to take
    off,
hover, and land.
"""

import time
from crazyswarm import Crazyswarm

def stage1_basic_flight():
    print("Stage 1: Initializing simulation...")
    swarm = Crazyswarm()
    allcfs = swarm.allcfs
    timeHelper = swarm.timeHelper

    # Takeoff: Fly to 1.0 meter over 2 seconds
    print("Taking off...")
```

```python
    allcfs.takeoff(targetHeight=1.0, duration=2.0)
    timeHelper.sleep(4.0)  # Wait for stabilization

    # Hover for 5 seconds
    print("Hovering...")
    timeHelper.sleep(5.0)

    # Land: Descend to 0.0 meters over 2 seconds
    print("Landing...")
    allcfs.land(targetHeight=0.0, duration=2.0)
    timeHelper.sleep(3.0)

    print("Stage 1 complete: Basic flight sequence finished.")

if __name__ == "__main__":
    stage1_basic_flight()
```

Listing 1: Stage 1: Basic Flight Control

2.3 Detailed Explanation

- **Environment Setup:**
  - Build your ROS workspace (typically in `ros_ws`) and source the setup file:
    ```bash
    source ros_ws/devel/setup.bash
    ```
  - Run the script in simulation mode using the `-sim` flag:
    ```bash
    python stage1_basic_flight.py --sim
    ```
- **Code Walkthrough:**
  - The script initializes the simulation via the `Crazyswarm` class.
  - It retrieves the list of drones (`allcfs`) and a timing helper (`timeHelper`).
  - Commands are issued to take off (to 1.0 m), hover, and then land (to 0.0 m), with delays ensuring each phase completes.
- **Observation:** In the simulation window, you should see the drone executing the flight sequence.

3 Stage 2: Trajectory Tracking (Single Drone)

3.1 Objective

Enhance Stage 1 by having the drone follow a smooth trajectory from its initial takeoff position to a specified goal position using the `goTo` command.

3.2 Boilerplate Code

```python
#!/usr/bin/env python
"""
Stage 2: Single Drone Trajectory Tracking
This script commands a single drone to take off, move to a new position using a
    smooth trajectory,
and then land.
"""

import time
from crazyswarm import Crazyswarm

def stage2_trajectory_tracking():
    print("Stage 2: Initializing simulation...")
    swarm = Crazyswarm()
    allcfs = swarm.allcfs
    timeHelper = swarm.timeHelper

    # Takeoff: Reach 1.0 meter altitude
    print("Taking off...")
    allcfs.takeoff(targetHeight=1.0, duration=2.0)
    timeHelper.sleep(4.0)

    # Trajectory: Move to a new position (e.g., [1.0, 1.0, 1.0])
    print("Executing trajectory (goTo)...")
    goal_position = [1.0, 1.0, 1.0]
    allcfs.goTo(goal=goal_position, yaw=0.0, duration=3.0)
    timeHelper.sleep(4.0)

    # Hover for 3 seconds at new position
    print("Hovering at new position...")
    timeHelper.sleep(3.0)

    # Land
    print("Landing...")
    allcfs.land(targetHeight=0.0, duration=2.0)
    timeHelper.sleep(3.0)

    print("Stage 2 complete: Trajectory tracking finished.")

if __name__ == "__main__":
    stage2_trajectory_tracking()
```

Listing 2: Stage 2: Trajectory Tracking

3.3 Detailed Explanation

- **Takeoff:** The drone is commanded to ascend to 1.0 meter in 2 seconds. A delay allows it to stabilize.
- **Trajectory Execution:** The `goTo` command calculates a smooth trajectory from the current position to `[1.0, 1.0, 1.0]`.
- **Hovering and Landing:** After reaching the target, the drone hovers briefly before landing.
- **Observation:** The simulation window should display the drone taking off, following the trajectory, hovering, and landing.

4 Stage 3: Coordinated Flight (Multiple Drones)

4.1 Objective

Extend your solution to control multiple Crazyflies simultaneously. Use group commands to synchronize maneuvers such as takeoff, formation flight, and landing.

4.2 Boilerplate Code

```python
#!/usr/bin/env python
"""
Stage 3: Coordinated Flight for Multiple Drones
This script commands multiple Crazyflies to take off, fly in a coordinated
    formation, and land simultaneously.
"""

import time
from crazyswarm import Crazyswarm

def stage3_coordinated_flight():
    print("Stage 3: Initializing simulation for multiple drones...")
    swarm = Crazyswarm()
    allcfs = swarm.allcfs
    timeHelper = swarm.timeHelper

    # Set all drones to the same group (e.g., group mask = 1)
    for cf in allcfs.crazyflies:
        cf.setGroupMask(1)

    # Synchronized Takeoff
    print("Coordinated takeoff...")
    allcfs.takeoff(targetHeight=1.0, duration=2.0)
    timeHelper.sleep(4.0)

    # Coordinated maneuver: Form a line formation
    print("Forming a line formation...")
    for cf in allcfs.crazyflies:
        offset = [0.5 * cf.id, 0, 0]
        current_pos = cf.position()
        goal = [current_pos[0] + offset[0], current_pos[1], current_pos[2]]
        cf.goTo(goal=goal, yaw=0.0, duration=3.0, relative=False, groupMask=1)
    timeHelper.sleep(5.0)

    # Synchronized Landing
    print("Coordinated landing...")
    allcfs.land(targetHeight=0.0, duration=2.0)
    timeHelper.sleep(3.0)

    print("Stage 3 complete: Coordinated flight achieved.")

if __name__ == "__main__":
    stage3_coordinated_flight()
```

Listing 3: Stage 3: Coordinated Flight

4.3 Detailed Explanation

- **Group Setup:** Each drone is assigned a group mask (1) so that broadcast commands affect all drones.
- **Synchronized Takeoff and Landing:** The entire swarm takes off and lands simultaneously.
- **Coordinated Maneuver:** The drones are commanded to form a line by moving to positions offset by their IDs.
- **Observation:** Verify in simulation that all drones take off, form the line, and land simultaneously.

5 Stage 4: Dynamic Trajectory Control

5.1 Objective

Implement dynamic re-planning for a single drone by commanding it to follow a circular trajectory. The drone's setpoints are continuously updated using `cmdPosition` to follow the circular path.

5.2 Boilerplate Code

```python
#!/usr/bin/env python
"""
Stage 4: Dynamic Trajectory and Re-planning
This script demonstrates advanced control where a single drone follows a circular
    trajectory
with continuous re-planning.
"""

import time
import numpy as np
from crazyswarm import Crazyswarm

def stage4_dynamic_trajectory():
    print("Stage 4: Initializing simulation for dynamic trajectory...")
    swarm = Crazyswarm()
    allcfs = swarm.allcfs
    timeHelper = swarm.timeHelper

    # Use the first Crazyflie
    cf = allcfs.crazyflies[0]

    # Takeoff
    print("Taking off...")
    cf.takeoff(targetHeight=1.0, duration=2.0)
    timeHelper.sleep(4.0)

    # Dynamic trajectory: circular motion in the XY plane
    print("Executing dynamic circular trajectory...")
    center = np.array([1.0, 1.0])
    radius = 0.5
    duration = 10.0
    steps = 50
    dt = duration / steps
    for i in range(steps):
        theta = 2 * np.pi * i / steps
        pos_x = center[0] + radius * np.cos(theta)
        pos_y = center[1] + radius * np.sin(theta)
        pos = [pos_x, pos_y, 1.0]
        cf.cmdPosition(pos, yaw=0.0)
        timeHelper.sleep(dt)

    # Hover and then land
    print("Hovering...")
    timeHelper.sleep(2.0)
    print("Landing...")
    cf.land(targetHeight=0.0, duration=2.0)
    timeHelper.sleep(3.0)
    print("Stage 4 complete: Dynamic trajectory executed.")

if __name__ == "__main__":
    stage4_dynamic_trajectory()
```

Listing 4: Stage 4: Dynamic Trajectory Control

5.3 Detailed Explanation

- **Takeoff:** The drone ascends to 1.0 meter and stabilizes.
- **Circular Trajectory:** A loop computes a circular path using numpy. The drone's position is updated continuously using `cmdPosition` to follow the circle.
- **Hover and Land:** After completing the circle, the drone hovers briefly before landing.
- **Observation:** In simulation, the drone should follow a smooth circular trajectory.

6 Stage 5: Advanced Swarm Behavior with Collision Avoidance

6.1 Objective

Develop an advanced control scenario where multiple drones perform coordinated flight while dynamically adjusting their trajectories to avoid collisions. This is achieved by computing avoidance offsets for each drone based on its neighbors and continuously updating setpoints using `cmdPosition`.

6.2 Boilerplate Code

```
#!/usr/bin/env python
"""
Stage 5: Advanced Swarm Behavior with Collision Avoidance
This script demonstrates advanced behavior where multiple Crazyflies execute a
    coordinated maneuver
and dynamically adjust trajectories to avoid simulated collisions.
"""

import time
import numpy as np
from crazyswarm import Crazyswarm

```

```python
def compute_avoidance_offset(current_pos, desired_pos, other_positions,
    min_distance=0.3):
    """
    Computes a simple repulsive offset to maintain a minimum distance from other
    drones.
    For each drone closer than min_distance, a repulsive force is applied
    proportional to the difference.
    """
    avoidance = np.array([0.0, 0.0, 0.0])
    for pos in other_positions:
        vec = np.array(desired_pos) - np.array(pos)
        distance = np.linalg.norm(vec)
        if distance < min_distance and distance > 0:
            avoidance += (vec / distance) * (min_distance - distance)
    return avoidance

def stage5_advanced_swarm():
    print("Stage 5: Initializing simulation for advanced swarm behavior...")
    swarm = Crazyswarm()
    allcfs = swarm.allcfs
    timeHelper = swarm.timeHelper

    # Set group mask for coordinated control
    for cf in allcfs.crazyflies:
        cf.setGroupMask(1)

    # Synchronized takeoff: All drones ascend to 1.0 meter.
    print("Coordinated takeoff for swarm...")
    allcfs.takeoff(targetHeight=1.0, duration=2.0)
    timeHelper.sleep(4.0)

    # Define target formation (example formation)
    target_formation = [
        [1.0, 1.0, 1.0],
        [1.2, 1.0, 1.0],
        [1.0, 1.2, 1.0],
        [0.8, 1.0, 1.0],
        [1.0, 0.8, 1.0]
    ]
    steps = 30
    dt = 0.5

    # Dynamic formation and collision avoidance loop
    for step in range(steps):
        for i, cf in enumerate(allcfs.crazyflies):
            target = target_formation[i % len(target_formation)]
            current_pos = np.array(cf.position())
            desired = np.array(target)
            error = desired - current_pos
            other_positions = [np.array(other_cf.position()) for j, other_cf in
    enumerate(allcfs.crazyflies) if j != i]
            avoidance = compute_avoidance_offset(current_pos, target,
    other_positions)
            new_setpoint = current_pos + 0.1 * error + avoidance
            cf.cmdPosition(new_setpoint.tolist(), yaw=0.0)
        timeHelper.sleep(dt)

    # Hover in formation for 2 seconds
    print("Hovering in formation...")
    timeHelper.sleep(2.0)

    # Synchronized landing: All drones descend to 0.0 meters.
    print("Coordinated landing...")
    allcfs.land(targetHeight=0.0, duration=2.0)
    timeHelper.sleep(3.0)
    print("Stage 5 complete: Advanced swarm behavior executed.")

if __name__ == "__main__":
    stage5_advanced_swarm()
```

Listing 5: Stage 5: Advanced Swarm Behavior with Collision Avoidance

6.3 Detailed Explanation

- **Group Coordination:** All drones are assigned a group mask (1) to ensure they receive synchronized takeoff and landing commands.
- **Synchronized Takeoff:** The swarm takes off simultaneously to 1.0 meter.
- **Dynamic Re-planning Loop:** For each time step:
  - Each drone calculates its error relative to a predefined target formation.
  - A collision avoidance offset is computed based on the positions of neighboring drones.
  - These factors are combined to update the drone's setpoint using `cmdPosition`, ensuring smooth and safe re-planning.
- **Synchronized Landing:** The swarm lands in unison after the maneuver.
- **Observation:** The simulation should display all drones taking off, dynamically adjusting their positions to form the target formation while avoiding collisions, and landing simultaneously.

7 Conclusion and Submission Guidelines

7.1 Final Submission Requirements

- A public Git repository containing all source code for the five stages.
- A detailed README with setup instructions, design rationale, and execution guidelines.
- A video demonstration (maximum 5 minutes) showing the simulation results for all five stages.

8 References

- Crazyswarm Documentation: `https://crazyswarm.readthedocs.io/`
- Python API Reference: `https://crazyswarm.readthedocs.io/en/latest/api.html`
- Crazyflie Firmware and CRTP Protocol: `https://www.bitcraze.io/documentation/`